МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Случайное БДП с рандомизацией

Вариант 11

Студент гр. 8304 _____ Барышев А.А.

Преподаватель _____ Фирсов К.В.

Санкт-Петербург

2019

Задание.

По заданному файлу F (типа file of *Elem*), все элементы которого различны, построить случайное БДП с рандомизацией. Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если входит, то удалить элемент *e* из структуры данных. Предусмотреть возможность повторного выполнения с другим элементом.

Цель работы.

Получить практический навык использования такой структуры данных, как случайное бинарное дерево поиска с рандомизацией. Ознакомиться с принципами структурной организации БДП, научиться применять функции интерфейса.

Функции и их описание.

Основные функции данной программы – это функция ввода:

node* Insert(node*, int);

И функция поиска и удаления элемента:

node* Remove(node*, int);

Ввод реализован с элементом рандомизации, чтобы снизить вероятность возникновения цепочки узлов, когда, к примеру, правая ветвь любого узла всегда нулевая. В этом случае, будет линейное время поиска нужного элемента, т.е. снижается эффективность применения данной структуры.

Данный вариант ввода приводит к тому, что из вводимого набора данных корнем может стать последний элемент. В противном случае, порядок ввода целиком определяет структуру.

Удаление реализовано просто: осуществляется рекурсивный обход дерева, причём с осуществляемой проверкой на то, является ли данный элемент меньше проверяемого узла или нет. Если меньше, то узел смещается влево, если больше – вправо. Далее, если удаляемого числа нет среди элементов БДП, то как только функция достигает листа, она вызывает себя в последний раз, т.к.

значение левого/правого поддерево – NULL. Таким образом, функция возвращает не изменённое БДП.

Тесты.

Данная программа тестировалась на различных наборах данных, ниже представлены результаты удаления элементов двух различных бинарных деревьев поиска:

A tree was introduced:

```
 12 63 93 124
      74
    18 27 55
        47
       24
  3 6
```

Remove the random element: 6

```
 12 63 93 124
     74
   18 27 55
       47
      24
  3
```

Remove the random element: 55

---

12 63 93 124

74

18 27 47

24

3

Remove the random element: 18

---

12 63 93 124

74

27 47

24

3

A tree was introduced:

47 81 93

63 75

8 17 18 24 35

7

6

1

Remove the random element: 93

---

47 81

63 75

8 17 18 24 35

7

6

1

Remove the random element: 63

---

47 81

75

8 17 18 24 35

7

6

1

Remove the random element: 6

---

47 81

75

8 17 18 24 35

7

1

Выводы.

Был Получен практический навык применения случайного бинарного дерева поиска с рандомизацией. Были уяснены принципы структурной организации БДП, получен опыт применения его интерфейса.

КОД ПРОГРАММЫ

```
#INCLUDE <IOSTREAM>
#INCLUDE <FSTREAM>
#INCLUDE <CSTDLIB>
#INCLUDE <CTIME>
#INCLUDE <STRING>

STRUCT NODE
{
      INT KEY;
      INT SIZE;
      NODE* LEFT;
      NODE* RIGHT;
      NODE(INT K){
            KEY = K;
            LEFT = RIGHT = 0;
            SIZE = 1;
      }
};

NODE* FIND(NODE*, INT);
NODE* INSERT(NODE*, INT);
INT GETSIZE(NODE* );
VOID FIXSIZE(NODE* );
NODE* ROTATERIGHT(NODE* );
NODE* ROTATELEFT(NODE* );
NODE* INSERTROOT(NODE*, INT);
NODE* JOIN(NODE*, NODE*);
NODE* REMOVE(NODE*, INT);

TYPEDEF NODE *BINSEARCHTREE;

NODE* FIND(NODE* P, INT K)
{
    IF( !P ) RETURN 0;
    IF( K == P->KEY )
        RETURN P;
    IF( K < P->KEY )
        RETURN FIND(P->LEFT,K);
    ELSE
```

```
        RETURN FIND(P->RIGHT,K);
}

INT GETSIZE(NODE* P)
{
      IF( !P ) RETURN 0;
      RETURN P->SIZE;
}

VOID FIXSIZE(NODE* P)
{
      P->SIZE = GETSIZE(P->LEFT)+GETSIZE(P->RIGHT)+1;
}

NODE* ROTATERIGHT(NODE* P)
{
      NODE* Q = P->LEFT;
      IF( !Q ) RETURN P;
      P->LEFT = Q->RIGHT;
      Q->RIGHT = P;
      Q->SIZE = P->SIZE;
      FIXSIZE(P);
      RETURN Q;
}

NODE* ROTATELEFT(NODE* Q)
{
      NODE* P = Q->RIGHT;
      IF( !P ) RETURN Q;
      Q->RIGHT = P->LEFT;
      P->LEFT = Q;
      P->SIZE = Q->SIZE;
      FIXSIZE(Q);
      RETURN P;
}

NODE* INSERTROOT(NODE* P, INT K)
{
      IF( !P ) RETURN NEW NODE(K);
      IF( K<P->KEY )
      {
            P->LEFT = INSERTROOT(P->LEFT,K);
```

```
            RETURN ROTATERIGHT(P);
      }
      ELSE
      {
            P->RIGHT = INSERTROOT(P->RIGHT,K);
            RETURN ROTATELEFT(P);
      }
}

NODE* INSERT(NODE* P, INT K)
{
      IF( !P ) RETURN NEW NODE(K);
      IF( RAND()%(P->SIZE+1) == 0 )
            RETURN INSERTROOT(P,K);
      IF( P->KEY>K )
            P->LEFT = INSERT(P->LEFT,K);
      ELSE
            P->RIGHT = INSERT(P->RIGHT,K);
    FIXSIZE(P);
      RETURN P;
}

NODE* JOIN(NODE* P, NODE* Q)
{
      IF( !P ) RETURN Q;
      IF( !Q ) RETURN P;
      IF( RAND()%(P->SIZE+Q->SIZE) < P->SIZE )
      {
            P->RIGHT = JOIN(P->RIGHT, Q);
            FIXSIZE(P);
            RETURN P;
      }
      ELSE
      {
            Q->LEFT = JOIN(P, Q->LEFT);
            FIXSIZE(Q);
            RETURN Q;
      }
}

NODE* REMOVE(NODE* P, INT KEY)
{
```

```
      IF(!P) RETURN P;
      IF(P->KEY == KEY)
      {
            NODE* Q = JOIN(P->LEFT, P->RIGHT);
            DELETE P;
            RETURN Q;
      }
      ELSE IF(KEY < P->KEY)
            P->LEFT = REMOVE(P->LEFT, KEY);
      ELSE
            P->RIGHT = REMOVE(P->RIGHT, KEY);
      RETURN P;
}

VOID DISPLAYBT(NODE* B, INT N, STD::OFSTREAM& FOUT){
            IF (B != NULL) {
                  FOUT << ' ' << B->KEY;
                  IF(B->RIGHT != NULL) {
                        DISPLAYBT(B->RIGHT, N + 1, FOUT);
                  }
                  ELSE FOUT << STD::ENDL;
                  IF(B->LEFT != NULL){
                        FOR (INT I = 1; I <= N; I++)
                              FOUT << "   ";
                        DISPLAYBT(B->LEFT, N + 1, FOUT);
                  }
            }
}

VOID DESTROY (NODE* &B)
      {     IF (B != NULL)    {
                  DESTROY (B->LEFT);
                  DESTROY (B->RIGHT);
                  DELETE B;
                  B = NULL;
            }
      }

INT MAIN(){
      SRAND(TIME(0));

      INT ARRAY[50];
```

```
            INT BUF[3];
            INT COUNTER = 0;
            STD::OFSTREAM FOUT;
            FOUT.OPEN("RESULT.TXT");
            STD::IFSTREAM FENTER("TESTDATA.TXT");
            BINSEARCHTREE H = NULL;
            INT BINTREEKEY;
            INT KEYFORREMOVE;
            BUF[0] = BUF[1] = 0;
            STD::STRING ENTERTEST;

            FOR(INT I = 0; I < ENTERTEST.SIZE(); I++){
                  IF((ENTERTEST[I]  <=  '9'  &&  ENTERTEST[I]  >=  '0')  ||
ENTERTEST[I] !=' '){
                        STD::COUT << "UNKNOWN SYMBOL '" << ENTERTEST[I] << "'"
<< STD::ENDL;
                        RETURN 0;
                  }
            }

            WHILE(FENTER >> BINTREEKEY && COUNTER < 50){
                  ARRAY[COUNTER] = BINTREEKEY;
                  COUNTER++;
                  H = INSERT(H, BINTREEKEY);
                  STD::COUT  <<  "DATA  WAS  RECEIVED:  "  <<  BINTREEKEY  <<
STD::ENDL;
            }
            STD::COUT << "ROOT NUMBER IS: " << ARRAY[0] << STD::ENDL;
            STD::COUT << "_____________________________________" << STD::ENDL;

            FOUT << "_____________________________________" << STD::ENDL << "A
TREE WAS INTRODUCED: " << STD::ENDL;
            DISPLAYBT(H, 1, FOUT);
            FOUT << "_____________________________________" << STD::ENDL;
            FOR(INT I = 0; I < 3; I++){
                  KEYFORREMOVE = ARRAY[RAND()%COUNTER];
                  WHILE(KEYFORREMOVE == ARRAY[0] || KEYFORREMOVE == BUF[0] ||
KEYFORREMOVE == BUF[1]){
                        STD::COUT  <<  BUF[0]  <<  "  "  <<  BUF[1]  <<  "  "  <<
KEYFORREMOVE << STD::ENDL;
                        KEYFORREMOVE = ARRAY[RAND()%COUNTER];
```

```
                    STD::COUT << BUF[0] << " " << BUF[1] << " " <<
KEYFORREMOVE << STD::ENDL;
                }
                BUF[I] = KEYFORREMOVE;
                FOUT << "REMOVE THE RANDOM ELEMENT: " << KEYFORREMOVE <<
STD::ENDL;
                REMOVE(H, KEYFORREMOVE);
                FOUT << "_____________________________________" << STD::ENDL;
                DISPLAYBT(H, 1, FOUT);
                FOUT << "_____________________________________" << STD::ENDL;
            }


            DESTROY(H);
            FENTER.CLOSE();
            FOUT.CLOSE();
            RETURN 0;
      }
```